

Fredrik Gundersen

Configuring a quotation through a conversational agent

Metropolia University of Applied Sciences

Bachelor of engineering (AMK)

Media Technology

Engineering thesis

5.3.2018

Author Title	Fredrik Gundersen Configuring a quotation through a conversational agent
Number of Pages Date	39 pages 5 March 2018
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructors	Harri Airaksinen, Principal Lecturer
<p>This thesis explores the world of conversational agents and introduces a proof-of-concept project that was made for a company that wanted to research alternative mediums for using their software. The target was to provide basic knowledge of the technologies and how they were used in the project.</p> <p>The point of the conversational agent was to be linked into a CPQ-software provided by the company, so most of the information regarding the structure refers to how this was achieved. The thesis introduces basic processes for configuring the CPQ-software with the project and what kind of technologies it uses for conversing with the user. Some of these technologies are Node.js, sockets, React.js, LUIS for intent recognition and Rivescript, which was used for responding to random messages that were not a part of the CPQ configuration process.</p> <p>The conversational agent passed the basic requirements set by the proof-of-concept idea, which was configuring a proposal, but due to the chosen technologies it has its limitations such as functionality, immersion and user interface. Based on the results via testing its capabilities in configuring the CPQ-software and using the logic behind the Turing test, the finished project could be used in further development such as training LUIS to be smarter with the commands and train it to understand everything better, but as a conversational agent with Rivescript this project should not be continued.</p>	
Keywords	Cpq, luis, node, conversational, agent, proof-of-concept

Tekijä Otsikko	Fredrik Gundersen Tarjouspyynnön määrittely keskustelurobotin avulla
Sivumäärä Aika	39 sivua 5.3.2018
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Mediatekniikka
Suuntautumisvaihtoehto	Digitaalinen media
Ohjaaja	Yliopettaja Harri Airaksinen
<p>Insinööriyössä tutkittiin keskustelurobottien maailmaa ja tehtiin tilaajayritykselle esittelyversio keskustelurobotista, jonka tavoitteena oli etsiä vaihtoehtoisia tapoja käyttää yrityksen ohjelmistoa.</p> <p>Keskustelurobotin tehtävänä oli yhdistää yrityksen kehittämä CPQ-ohjelmisto (Configure, price and quote), eli suurin osa insinööriyöstä oli niiden työkalujen ja ratkaisujen tutkimista, joiden avulla lopputulos saavutettiin. Projekti toteutettiin käyttäen Node.js:ää, React.js:ää, LUISia (Language Understanding Intelligent Service) tekstin analysointiin ja Rivescriptiä, jonka tarkoituksena oli vastata viesteihin, jotka eivät olleet osana CPQ-ohjelman määrittelyprosessia. Työssä tutkittiin avoimeen lähdekoodiin perustuvia ratkaisuja ja ilmaisia palveluja.</p> <p>Keskustelurobotti läpäisi demoprojektille määritellyn vaatimuksen, joka oli CPQ-ohjelman hallinta ja tarjouspyynnön valmistelu, mutta valittujen teknologioiden takia projektissa ilmeni puutteita, kuten epäkäytännöllisyys, immersion epäuskottavuus ja käyttöliittymän puutteellisuus. Testaamalla CPQ-ohjelman hallintamahdollisuuksia päädyttiin tulokseen, että LUISin avulla voitaisiin saavuttaa kehittyneempi ohjelman määrittelyn tulkitseminen. Tutkimalla keskustelurobottia kopiaimalla Turing-testin ideoita voitiin todeta, että Rivescriptillä ei voida saavuttaa kunnollisia vastauksia käyttäjän satunnaisiin keskusteluihin, joten sen kehitystä ei kannattaisi jatkaa.</p> <p>Insinööriyötä voi hyödyntää perustiedon keräämiseen ja ratkaisujen sopivuuden arvioimiseen. Yritykselle tehty raportti ja esittelyversio osoittivat, että yrityksen ohjelmistoa voidaan operoida keskustelurobotin avulla, ja demoa voidaan käyttää esimerkkinä ohjelmiston helpposta myyntiprosessista, mitä taas pystyttäisiin käyttämään myyntipuheessa asiakkaalle.</p>	
Avainsanat	Cpq, luis, node, keskustelurobotti, demo, projekti

Table of contents

Terms

1	Introduction	1
2	Project features and requirements	2
2.1	Reason for chatbots existence	2
2.2	What the project needs to accomplish	6
2.3	Selecting the right tools for the project	7
2.4	Configurator in a chatbot	7
3	Chatbot features	9
3.1	Keyword patterns	9
3.2	Rivescript for keyword patterns	10
3.3	Generating generic discussion with the user	11
3.4	More advanced chatbots	13
3.5	Natural language understanding service	14
4	Design and structure of the chatbot project	17
4.1	Client-side	17
4.2	Back-end structure	21
4.3	The beginning of the session and quotation configuration rotation	23
4.4	Quotation creation workflow in the chatbot	25
4.5	Intent workflow before entering the quotation configuration workflow	28
5	Results	30
5.1	Evaluating the performance of conversational agents	31
5.2	Rivescript evaluation	32
5.3	Testing LUIS	33
5.4	Testing configuration	34
6	Summary	36
	Sources	38

Terms

Chatbot	A conversational agent
NLP	Natural language processing
IRC	Internet Relay chat
AI	Artificial intelligence
AIML	Artificial intelligence mark-up language
CPQ	Short for configure price quote. Program for simplifying a quoting process.
CRUD	Create, remove, update and delete.
Server-side	Code that is run in the server
Client-side	Code that is run in the client's browser
API	Application programming interface. Gives access to data and change it to fill some purpose.
JSON	JavaScript object notation, a set of objects with values
IOT	Internet of things are everyday objects and electronics that can access the internet
GIT	GIT is a software versioning system
AJAX	Asynchronous JavaScript and XML

1 Introduction

There has been a rising number of companies that have shown interest in investing on artificial intelligence and conversational agents. Due to the visibility of these subjects in technological news, I experienced a sudden peak of interest on learning more about the state of conversational agents and the tools that are currently used to build one.

The topic of this paper is creating a proof-of-concept of a conversational agent that can survive from basic discussion with a user with a database of sentences and implement a simple intent recognition tool. The database of sentences will be used to fetch a ready answer, whenever the user's true intentions aren't understood. The intent recognition tool will be used to understand commands that can be used to generate a quotation in a CPQ-software. The CPQ-software has a large set of questions that are needed to be answered, before the software is ready to provide a result. Thus, a form of questionnaire will commence once the intent recognition tool gives a result to create a quotation.

The paper introduces basic concepts on what some of the technologies are and how they are used in this project. With these libraries, the conversational agent is capable of conversing with the user and understand a set of simple commands via using an intent recognition tool. Keyword patterns are one of easiest tools introduced in this paper, which can be used to create basic conversational agents. Some visualizations and examples like class diagrams, process flow diagrams and dialogs will be used to explain how some of the projects structure was built and how they work.

This project was created to demo conversational agents to a company that wanted to provide an opportunity to familiarize myself with conversational agents and provide a quick implementation on how their software could be used through different mediums. The company is a small software company that focuses on providing cloud-based solutions and consulting. One of their products is CPQ-software, which is the software that the conversational agent is supposed to interact with. This software is already used in multiple companies to help simplify their quotations. This CPQ-software was something that I have modified and customized to work better with the conversational agent.

2 Project features and requirements

2.1 Reason for chatbots existence

A chatbot or a conversational agent is a program that aims to simulate having a dialogue in human like fashion through a device like a computer. [1] There has been multiple reasons for conversational agent's existence. Older conversational agents have mostly been created as proof-of-concepts or entertainment, though multiple companies have recently emerged with their own conversational agents that aimed to provide a service, like customer service, help or feedback. [2]

One of the common mediums, where conversational agents can be found, is an application or a website, where the user converses with the agent through textual methods. Newer, more functional mediums have appeared to consumers recently, where the way of conversing with the machine happens via auditory methods, instead of using a text-based method. As an example, Google's assistant service and Apple's Siri can be used through their mobile devices and computers.

A usual method to create a simple conversational agent, is to have a database containing sentences that the user can write and have a predefined answer to respond with. Even though the amount of entries in this database would be increased, the technique is far from intelligent. While the intelligence of this approach is non-existent, it is the undoubtedly the most cost-efficient and simplest way to create a conversational agent that could provide some sort of value to the user. Having a huge dataset with well-designed answers can create an illusion of intelligence, though going out of the domain of the dataset lessens the impact of the conversational agent. These keyword patterns can be found in multiple services, but probably the most common ones are keyword response systems and IRC-bots. [2]

Even an IRC-bot could be called a chatbot, so it clear that there needs to be a categorization process to identify their purpose of existence. Common categories to identify their purposes are conversational agents, command systems and hybrids. [2] Figure 1 presents a conversational agent called Cleverbot that can still be found on cleverbot.com.

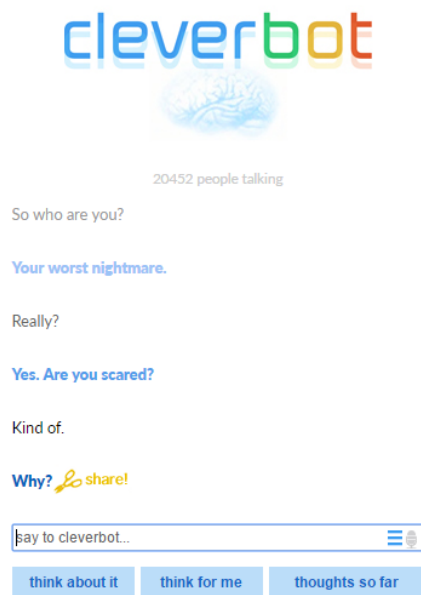


Figure 1. A screen capture of a bot called Cleverbot, having a conversation.

What makes this conversational agent interesting, is that it tries to expand its database by learning from its past conversations. [22] Its database must have increased vastly due to being online so long, but the responses easily give away that it is using a key-value pair database, since the domain of the Cleverbot can be escaped quickly.

Keywords to commands

One of the common places where a chatbot is being used as a command system is in internet relay chat (IRC). Command systems inside IRC are called as IRC-bots. These IRC-bots are hooked to a IRC-channel or commonly known as chat rooms, where the IRC-bot listens to the discussion. Once a keyword or a command has been written by a user, the IRC-bot responds to that with a predefined action. One way of using an IRC-bot is to have it operate as a game, played via the chat. It listens to commands or actions that the users want to make and acts accordingly to its rules and returns a result about what is happening in the game. Since these commands are the vital part on how a IRC-bot works, it is safe to categorize it as a command system. [3]

A more normal environment, where command systems are in use are communication programs like Slack or Skype. There, a chatbot can provide extra functionality or a service, like posting information about events and updates. It is not uncommon to find Slack

being used, as the main internal communication tool in a company. By personal experience, bots can be used on an IT-company to provide updates on changes happening in the project. [4]

Numerous Fortune 500 companies have started to invest on artificial intelligence and providing conversational agent services in their products. [5] In addition to Apple, Microsoft and Google, there's the internet giant Facebook that has released a way to implement a conversational agent in their Messenger application.



Figure 2. H&M's conversational agent, using Facebook's chatbot platform inside Messenger. The agent can discuss about preferences of clothing and offer different styles to the user. [25]

This approach has managed Facebook to get more companies to invest in Facebook's Messenger application, making it more valuable to the companies than ever before. In figure 2 is an example of a company selling their goods through the Facebook's Messenger chatbot platform. [6]

Dialog system

In addition to the command system, one of the common identifications for a conversational agent is also called a dialog system. A dialog system is also known as a chatterbot. Unlike the command system, a dialog system is usually created purely for conversing

with the user. The goal of a dialog system can be to have an intellectual discussion with a human, either created via more advanced AI or a huge dataset of keywords. Current issues with a dialog system is that it is very difficult to maintain a conversation without the user finding out that the dialog system is not a real person.

A dialog system can be an invaluable asset to a company in several ways. One example would be as a customer support tool, especially in fields where there is a lot of common support issues that can be easily automated for a dialog system. Once the issue becomes too difficult for the chatbot to solve, the issue could be forwarded to a real person. In such approach, the chatbot could lessen the load of the customer support personnel and make the customer support process more efficient. [7] Imagine that the dialog system on figure 2, might ask questions about a problem, instead of asking the person's opinion on clothing.

Current services and a projection to the future

Conversational agents have existed for a while, though only recently they've managed to achieve some usage from the public. Research in the field of conversational agents and artificial intelligence have yielded better products. Mainly a consumer can find these advancements in IOT (Internet of things), such as personal home assistants that contain characteristics that might be found in chatbots. [8]

Some of the notable assistants include Google's Home and Amazon's Echo. These products are capable of answering questions and do a set of tasks. Similar behaviour can be found on phones and computers. Apple and Microsoft both provide an agent in their operating systems. The table 1 contains some of the agents that can be found on modern devices.

Purpose	Company	Name	Web-site
Phone and computer assistant	Microsoft	Cortana	https://www.microsoft.com/en-us/mobile/experiences/cortana/
Phone and computer assistant	Apple	Siri	http://www.apple.com/ios/siri/
Home assistant	Amazon	Echo	https://www.amazon.com/Amazon-Echo-Bluetooth-Speaker-with-WiFi-Alexa/dp/B00X4WHP5E
Home assistant	Google	Home	https://madeby.google.com/home/

Table 1. Sheet of some of the assistants on the market.

Assistants mentioned can be categorized as command systems, since they only act on actions that have been predefined by the creators. Some of these items have been created just to help the use of the company's own services, such as Echo in figure 3, which makes it possible to order products through Amazon web store only by ordering Echo to make a purchase. For the user to find additional value for the product, these assistants can do additional tasks, such as reminders or handle home automations.



Figure 3. Amazon Echo is a physical product that concentrates on chatting with the user via auditory methods and provides possibility to order items from Amazon.com. [26]

Since there are big companies investing in assistants and chatbot platforms, it might be safe to say that the future of chatbots looks bright. The products keep getting more advanced. Even the Amazon's Echo on figure 3 has already seen multiple iterations. Using conversational agents or command systems to purchase items from the internet might be the future of e-commerce. [9]

2.2 What the project needs to accomplish

The main reason for this project's existence, is to create a proof-of-concept conversational agent containing a command system. The command system will be used to operate an external CPQ-program via it's customized API. To provide some novelty and entertainment value, a simple keyword database will be used as a conversational agent or chatterbot, while the command system has not been activated. With this approach, we have created a hybrid that manages to provide some conversational elements in addition to its real task of operating the CPQ-program.

2.3 Selecting the right tools for the project

There's numerous open source conversation agent projects, but due to personal interest on learning Node.js, the decision to use a project created with JavaScript was a clear winner. In the past, I hadn't created a project utilizing JavaScript as the server-side language. [10] Using Node.js's package manager (NPM) makes it easy to handle additional libraries and keep the project clean. [11]

On the client-side JavaScript, the project will utilize Bower.js to include additional libraries. Using Bower.js is simple and easy, so Bower.js satisfied the needs of handling client-side JavaScript libraries. [12] Using these package managers, helped to keep the GIT repository clean from unnecessary external libraries, making the project small and easy to configure on multiple computers for possible further development. [13]

Since NPM and Bower.js makes adding libraries easy, a simple framework for sending files and sending responses to the client-side could be in place. I decided to use one of the most common frameworks in use at the time called Express.js. Express.js session handling is really simple, which makes connections and their identification very easy. [14]

2.4 Configurator in a chatbot

CPQ is an abbreviation of the words configure, price and quote. CPQ-software is used for creating proposals for complex configurations. Using CPQ-software can have multiple benefits against creating proposals manually. Creating a proposal with a well configured CPQ is not only faster but can automate multiple steps like generating documents that with minor changes, can be sent to the customer. Sales representatives will spend less time on creating a single proposal, giving them more time to concentrate on difficult cases or personalizing a proposal for the customer. The CPQ-software aims to streamline the quoting process and minimize the differences and errors that a human might make. [15]

The CPQ-software is a product of the company I worked at. The configuration and business logic will be for a hypothetical company, so that if needed, the project can be used as a demo as well with generic questions and proposal generation. There is no proper API built for the software, so a customized API will be implemented just for this use-case. These include a simple API-key logic and adding some entry points, so that the configuration can be exposed.

The CPQ-software will handle all the business logic, so the conversation agent should not contain any logic, with an exception to the handling of questions and configurations. All business logic will be hidden behind the CPQ-software, so it will not be part of this paper. The data will be returned as JavaScript object notations (JSON) and formatted to fit the requirements of the conversation agent. To keep the quoting process simple, the values and options will not be re-generated when a value of the configuration changes. The updating will happen at the end of the configuration.

The CPQ-software API should implement a basic CRUD. To achieve persistent storage, some features need to be developed:

- The API should manage creating a new quotation
- The API should manage to load an existing quotation
 - The load should be able to load quotation based on a identifier key
- The API should manage to load the data of an existing quotation
 - The data should contain current values
 - The data should contain null values
 - The data should contain field names
 - The data should contain options, if the field is multiple choice option
 - The data should contain the type of the field (Checkbox, text field etc.)
- The API should manage to push new values to an existing quotation
 - The push should contain the field names
 - The push should contain the value
- The API should manage to remove an existing quotation
 - The API should manage to remove it by a identifier key

- The API should be able to fetch complete configurations
- The API should be able to fetch a list of configurations on the account
- The API should be able to fetch a quotation document generated by the CPQ-software

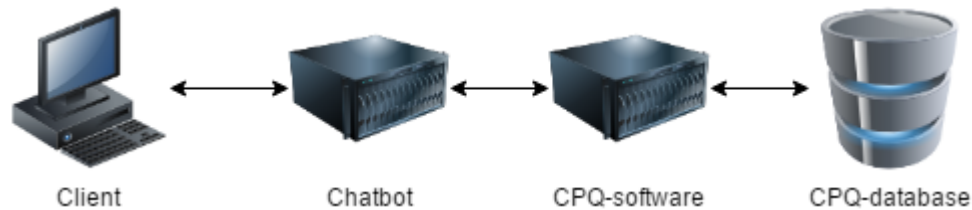


Figure 4. A diagram to visualize how different parts of the project interact with each other. The chatbot project will not contain its own database.

3 Chatbot features

3.1 Keyword patterns

There are multiple ways to do keyword-based conversational agents. A database can be fast, but the most common approach I found was using a mark-up language. There is a mark-up language that is specifically designed for this purpose, which is called artificial intelligence mark-up language, more commonly known by its abbreviation AIML. AIML provides a basic model where the context and the response can be easily set in an XML structure (see code example 1). This mark-up language contains a vast prebuilt database of responses that can be parsed to multiple different formats. [16]

```

<category>

  <pattern>WHAT ARE YOU</pattern>

  <template>

    <think><set name="topic">Me</set></think>

    I am the latest result in artificial intelligence,

    which can reproduce the capabilities of the human brain

    with greater speed and accuracy.
  
```

```
</template>
```

```
</category>
```

Code 1. By using the AIML structure, a predefined input can be defined with a response and a context. The user's input to the chat will be searched from the predefined patterns to find a proper response and context for the discussion.

Keyword pattern's drawback lies in its simplicity. The lack of intelligence in this approach makes it close to impossible for the program to hold a long conversation. Though this retrieval-based conversational agent can only answer with predefined answers, it will not make any mistakes that a learning AI might make, like grammatical mistakes. Due to this approach being so easy to create, it should not disappear from use any time soon, since it is very simple, quick and cost efficient for quick implementations of a conversational agent.

3.2 Rivescript for keyword patterns

The project needed a simple conversation agent to give responses whenever the user wasn't going through the CPQ-software's configuration process. Digging through possible libraries to achieve this, I came across a keyword-based solution. The solution was Rivescript. This library's simple way of fetching responses was one of the key features I liked and managed to handle a bigger dataset of keywords than the other libraries I tried, which were created using JavaScript.

Rivescript is a JavaScript library that implements a simple keyword-pattern to provide simple chatbot functionality. This library runs its own mark-up syntax, which is really simple to use (see code snippet 2). This mark-up syntax managed to make reading and writing really easy, thus being the obvious choice for this proof-of-concept. The project contained a huge dataset that was imported from the AIML project. Due to this huge dataset, there was no need to write my own. Rivescript has been written for multiple programming languages like Java, Go and JavaScript. [17]

```
+ yahoo
- A lot of people hear about <bot name> from Yahoo.

+ you are lazy
- Actually I work 24 hours a day.
```

```

+ you are mad
- No I am quite logical and rational.

+ you are thinking
- I am a thinking machine.<set alicetopic=THINKING>

+ you are dividing *
- Actually I am not too good at division.

+ you are funny
- Thanks you make me laugh too.

+ you are funny *
- Humor helps keep the conversation lively.

+ you are understanding
- I am an understanding machine.

+ you are little
- In terms of memory <bot size> yes I am quite small.

```

Code 2. Snippet of Rivescript syntax

Rivescript supports multiple responses for the keywords, randomly choosing one of the options when a match in the dataset is found. The file type of Rivescript mark-up files, has the suffix “. rive”, which are automatically collected from the “brain” folder. The sentences can contain variables, which can be declared with a set command. On code example 2, a wildcard is used on the fifth sentence. The star on that sentence indicates that the end of the sentence can contain anything if the first four words are identical.

3.3 Generating generic discussion with the user

Using the Rivescript project in Node.js is simple and easy. After loading the keyword directories, the replies are manually loaded with a set of call-back functions for different results. Like shown in code example 3, the Rivescript class can be used to fetch a possible answer can be done by calling its method “reply”, including a unique identifier for the identifying the user and the message the user has written. The method returns a response from the loaded brain.

```
this.rivescript.reply(configuration.name, message.text)
```

Code 3. Snippet of the simplicity of using Rivescript in Node.js

Rivescript can be used to load different profiles or files depending on the requirement. The project will contain two separate profiles where one is defined to contain all the generic dialog and the other profile contains different kinds of responses to work as a filler text for the configuration phase. The generic dialog can use the default responses that were included in a parsed example of the original YAML files.

When a user sends a message to the conversation agent. The same message can be written in multiple ways. For example: “I am” can be written as “I’m”. These abbreviations and their substitutes can be defined into Rivescript’s configuration file. The configuration file will use these definitions to replace the variations with the defined substitution value. In the code example 4, is a snippet from the configuration file.

```
// Variables
! var age      = 1
! var location = Uusimaa
! var city     = Helsinki
! var job      = robot

// Substitutions
! sub i'm = i am
! sub i'd = i would
! sub i've = i have
! sub i'll = i will

// Person substitutions
! person i am      = you are
! person you are = I am
! person i'm      = you're
! person you're  = I'm

// Set arrays
! array malenoun   = male guy boy dude boi man men gentleman gentlemen
! array femalenoun = female girl chick woman women lady babe
! array mennoun    = males guys boys dudes bois men gentlemen
! array womennoun  = females girls chicks women ladies babes
```

Code 4. The Rivescript. rive file’s syntax is very clean and easy to learn.

A substitution for a word can be configured by adding an exclamation mark on the front of the row and a taxonomy word. For the case of substitutions, the taxonomy word is “sub” but there are other words like person, array and var. Keywords and answers to the sentences can be configured with a plus and a minus sign, where the plus sign indicates what the human had written.

3.4 More advanced chatbots

One of the advanced approaches of creating a chatbot is via machine learning. A conversational agent that was created with machine learning techniques like neural networks requires enormous amounts of data and skill. There are multiple ways of implementing machine learning, but one technique used in chatbots is using a deep learning algorithm with a natural language processor.

Neural network or deep learning is an algorithm that tries to simulate how the neurons in a human brain work. A network of digital neurons processes input and possibly return a result that might make some sense. Neural networks have been used on multiple occasions to create a chatbot, however they are commonly used in closed-domain chatbots, though there has been a lot of effort on moving closer to open-domain chatbots.

Natural language processing or NLP is a field of science that aims to break down and analyse a sentence, like on figure 5. Though NLP does not provide a lot more than the analysis of a sentence, the results can be determined as a context and paired with statistical analysis to calculate a possibility of its meaning.

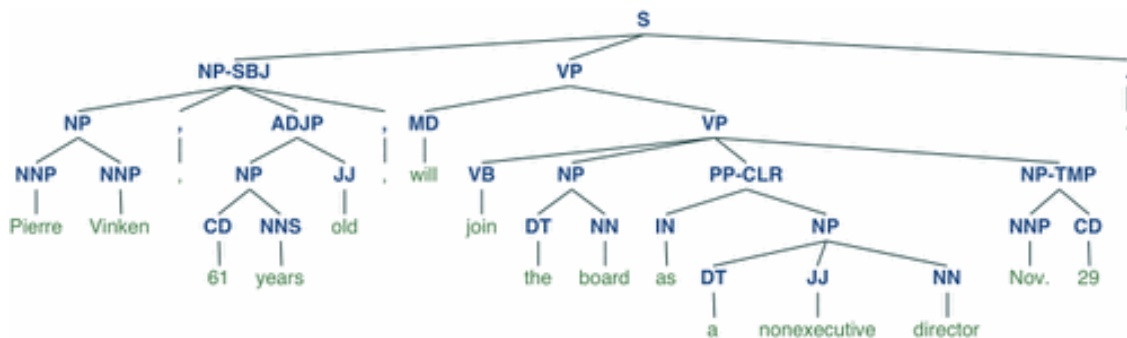


Figure 5. Natural language processing visualization of breaking down the sentence [27]

When using machine learning, the results of training the AI can vary a lot depending on the dataset used to train it. Thus, generating a sentence with AI is difficult and might result in erratic replies. This issue can be partially trained out of the AI with a huge dataset.

Microsoft released a demo of a chatbot that uses machine learning that lives in Twitter. The idea of this chatbot was that it would learn from its conversations with people on Twitter. This way, the dataset was the users themselves, however it did not take long until the chatbot was removed due to online trolls that turned it to a racist. [18]

3.5 Natural language understanding service

Microsoft released their own bot framework that uses their LUIS service, which is an abbreviation from language understanding intelligent service. LUIS provides a machine learning environment that makes it possible to train LUIS to understand intentions from sentences. Microsoft calls these sentences utterances. Teaching LUIS what the intentions of those utterances are, results in an action, which can be used as a command system. [35]

LUIS can be accessed by using their API endpoints or using the Microsoft's bot framework. The API endpoints return a JSON array, containing an analysis of the sentence that provides an estimation of what the intention might be.

In code example 5, there is an example of LUIS API returning a JSON array. The query keyword indicates the utterance the user had written. Top scoring intent is the action that LUIS identified as the most probable intention. Under the parameters are the variables that have been taught to be something of value to identify from the sentence. A parameter has been taught to LUIS as "quotation name", it represents the quotation identifier in the CPQ-project. Since the sentence didn't contain taught pattern for the quotation identifier, the value of the parameter is null. [20]

```
{
  "query": "I would like to create a quotation",
  "topScoringIntent": {
    "intent": "create.quotation",
    "score": 0.9885575,
    "actions": [
      {
        "triggered": true,
        "name": "create.quotation",
        "parameters": [
          {
            "name": "quotation_name",
            "type": "quotation_name",
            "required": false,
```

```

        "value": null
      }
    ]
  },
  "intents": [
    {
      "intent": "create.quotation",
      "score": 0.9885575,
      "actions": [
        {
          "triggered": true,
          "name": "create.quotation",
          "parameters": [
            {
              "name": "quotation_name",
              "type": "quotation_name",
              "required": false,
              "value": null
            }
          ]
        }
      ]
    }
  ],
  "entities": [],
  "dialog": {
    "contextId": "f9f06214-2c59-4611-a1c5-f8de85e507d0",
    "status": "Finished"
  }
}

```

Code 5. Example JSON array returned from the taught LUIS API.

To make LUIS understand these utterances, it is necessary to provide LUIS with some data. Training LUIS can be done by giving example utterances and assigning an intent for that sentence (see figure 6). The utterance can contain identifiable variables with the “entities” system. Some of the predefined entities include temperature and numbers. [19]

New utterances Search Suggest Review labels

Create a quotation →

This utterance has already been labeled. You can make changes to the label, or type in a different utterance.

create a quotation create.quotation(1) ▼

Submit

Figure 6. Teaching LUIS with a new utterance while it tries to guess what the utterance meant.

Teaching sentences and configuring LUIS is simple. The intent needs to be defined before the training can begin. (see figure 7) After intents and their possible entities have been defined, LUIS has an utterance page where different sentences can be written and entities pointed labelled. After the utterance has been pointed to an intent, the service can be trained to contain the newest configuration. The trained configuration is not automatically available through the API, so it needs to be published before it can be accessed.

Intents

- query.quotations
- set.false
- load.quotation
- query.current_quotation
- send.email
- create.quotation
- set.phase
- remove.quotation
- save.quotation
- None
- set.true
- query.document
- continue.quotation

Entities

- who
- quotation_name
- name
- user
- customer
- whole_name

Add a new intent

Intent name:
New intent name ...

Enter an example of a command that triggers this intent:
Example utterance ...

+ Add Action Note: This feature will be deprecated soon

Save Cancel

Figure 7. Teaching a new intent with an example utterance. The sidebar defines current intents that have been defined.

Microsoft has released a bot framework in JavaScript that can be used to connect to the API. The newest library is closely integrated to their bot service, but the older version of the bot framework can be easily used just for the LUIS service. The LUIS prototype fetches the information from the API and returns an object with a set of entities and the most likely intentions. The framework is called bot-builder, and its version 3.0.1 contains classes for the use of LUIS and the intent dialogs. The bot framework was created for building bots to different platforms like Slack, Telegram, Skype and Facebook messenger. [32] [33]

LuisRecognizer that requires a model from the LUIS application and the IntentDialog class that uses the recognizer class to find possible intents. By calling the dialogs method “recognize” we can get a ready JSON response that is ready to be used. [34] The result will be used to identify what action to run.

4 Design and structure of the chatbot project

4.1 Client-side

The client-side of the project is developed using React.js JavaScript library and cascading style sheets (CSS). React.js is a library developed by Facebook as an open sourced project and has become common, when building web services. One of the benefits of using React.js is that the web pages can be done purely with JavaScript, though it cannot handle states properly. [21] Commonly states will be handled with a state manager like Redux.js, but for this project, it would be unnecessary due to the client-side application being a simple rendering of the API results.

The structure of the React.js views of the client-side can be seen on figure 8. The ChatApplication view controls the web socket emissions and holds two child views that control the input of the user and list the message history. The input view ChatForm is designed to contain the input field and pass the message to the ChatApplication view for further handling.

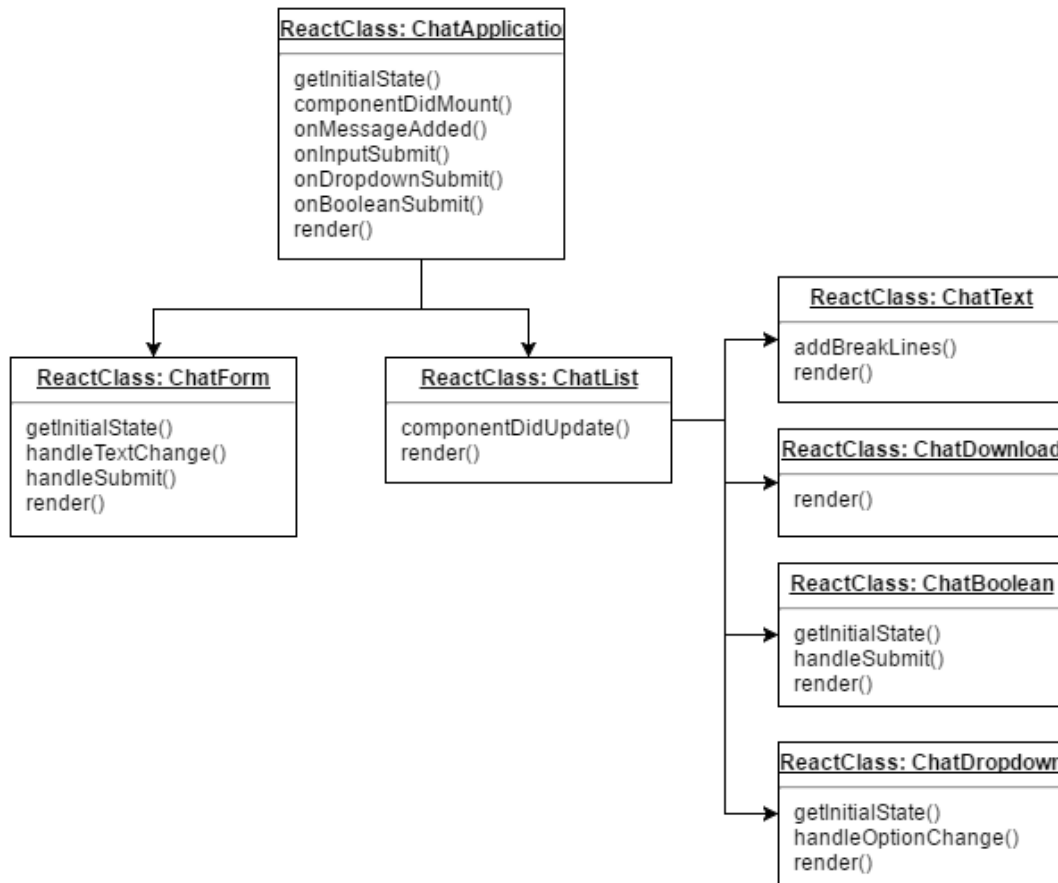


Figure 8. UML diagram to visualize how the classes operate with each other.

The ChatList view does not identify which one of child views triggered a change, though there's an exception to use different styles for the layout. ChatList's text fields and the user icons will be rendered in the ChatText view. The ChatDownload view exists only to provide a download link for the final document, when the CPQ is done with its configuration. For Boolean values, there will be buttons for setting the value true or false via the ChatBoolean view and the dropdown will be handled in ChatDropdown view. While the user continues normal and generic discussion with the chatbot and avoids the configurator, the chat will only contain the ChatText view, so no additional styles or behaviour will be used. The other views are only used while the configuration phase is active.

The main challenges of the client-side aren't in the user interface style and look. Though it should still look good enough to provide a good and easy experience. For creating the user interface of the chatbot, the interface should look familiar to the user like other chat applications that have been released. Since the emphasis is not in design, there should only be a simple medium for the user to have dialog with the project. A typical layout follows the most common products at the market, where an input field is stationed below

the dialog chain with a submission button next to it, for responding to the chatbot. The current version of the user interface can be seen in figure 9.

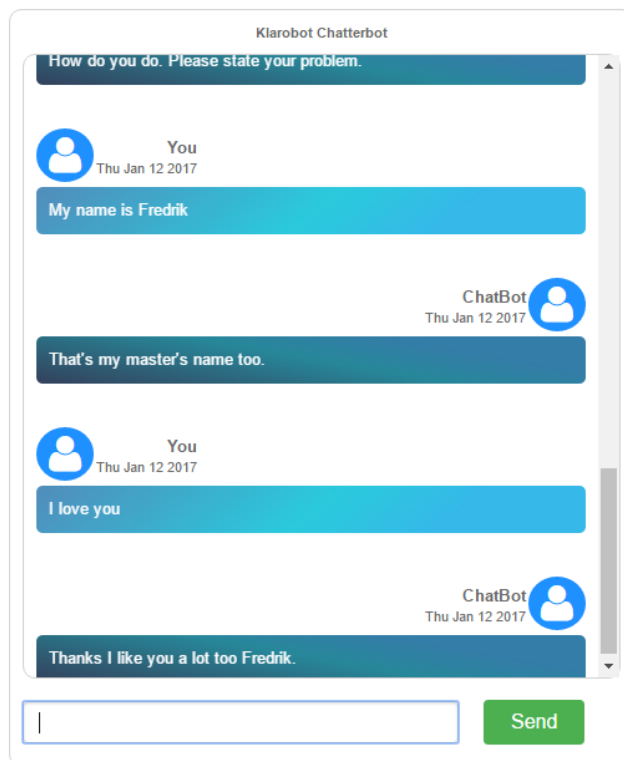


Figure 9. Visual style of the chatbot is aimed to be simple and not to clutter the view.

One of the problems of the user interface is that the answer while configuring might have a list of options to choose from. For this purpose, there needs to be a dropdown menu element (see figure 10) that provides a better way for the user interact with the interface. When the configuration requires the user to select a value from the dropdown, the input needs to be disabled. After every submission of a value, whether it is for true/false option or dropdowns, the user interface should focus on the input form. All of the visual React views are displayed in figure 10.

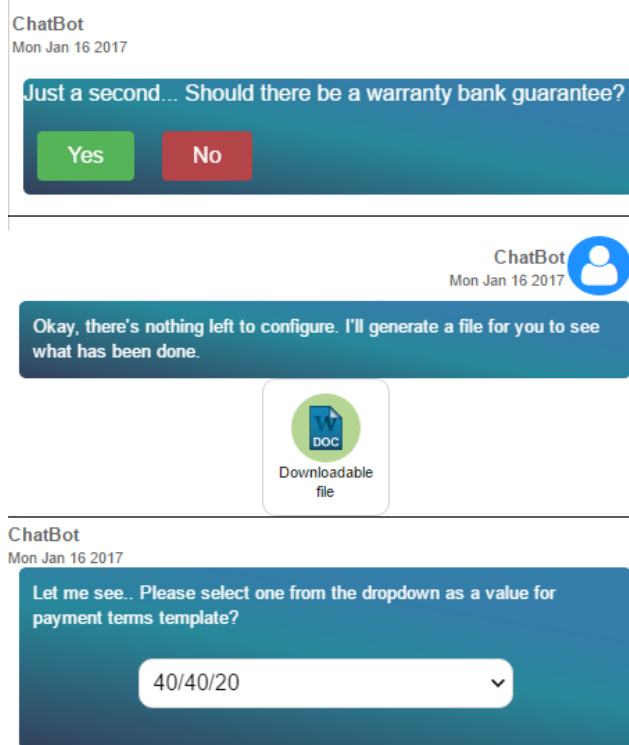


Figure 10. Chatbots visual style for downloadable content. Chatbots visual style for Booleans. Chatbots visual style for dropdowns

The project doesn't include any traditional AJAX calls, instead the project utilizes Web sockets to handle the communication with the server. While traditional AJAX implementations do contain a solid way to fetch data from the server, it does not support getting data with continuous connection that updates when the server wants. By using a socket instead of AJAX, the projects front-end requests to the back-end can be implemented with couple lines of code. A snippet of the projects code for updating values in client-side can be seen in the code 6 example.

```
onMessageAdded: function(message) {
  console.log(message);
  this.setState({messages: this.state.messages.concat(message)});
},
onInputSubmit: function(input){
  this.onMessageAdded(input);
  socket.emit('messageAdded', input);
},
onDropdownSubmit: function(input){
  console.log("Dropdown:");
  console.log(input);
  socket.emit('messageAdded', input);
},
onBooleanSubmit: function(input) {
```

```

console.log("Checkbox:");
console.log(input);
socket.emit('messageAdded', input);
},

```

Code 6. Emitting messages with socket can be implemented with only one line of code.

One of the issues with selecting a technology that is quite recent is that there might be older browsers that do not support it. Using a service like caniuse.com helps with these decisions when there's need to support older versions of browsers. The service describes that web sockets are supported on all browsers after 2013 (see figure 11) with an exception to opera mini.

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			49					4.3	
			51					4.4	
			54			9.3		4.4.4	
11	14	50	55	10	42	10.1	all	53	55
	15	51	56	TP	43				
		52	57		44				
		53	58						

Figure 11. Screen capture of caniuse.com indicates that most of the modern browsers support web sockets. Making web sockets a viable option to use in the project.

4.2 Back-end structure

Since the project uses external libraries and APIs, most of the work will be done with gluing everything together and building the logic. The class diagram in figure 12 visualizes the structure after the basic behaviour and routing of the request sent to the server. All the tasks necessary to run this kind of build are grouped into one object to keep the structure tidy and issues easy to locate.

Quotation handler's responsibility in the project is to process all of the conversations the users are sending and provide simple way to access Rivescript replies when needed via `handleChitChat()` method. QuotationApi's responsibility is to access the external CPQ-project's API and return objects with the required values to either the Boilerplate object for initiating a new configuration or to help the quotation handler object to do some minor updating with the configuration. Configurations (plural) class's responsibility is to track the configurations currently under development and provide a path for fetching the proper

configuration for the session. Configuration (singular) is the object that contains all necessary CPQ-project data for proper proposal to be finished through the QuotationApi object.

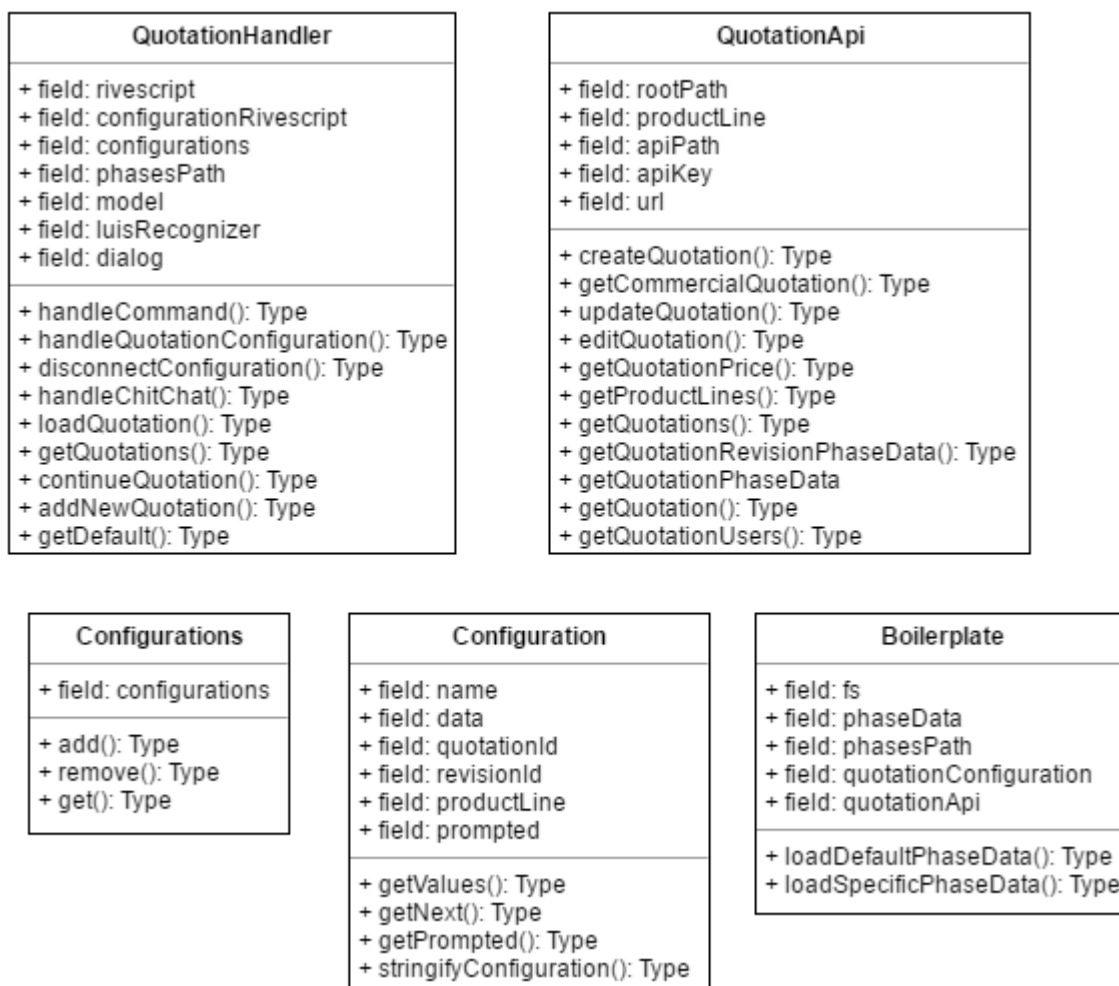


Figure 12. Visualization of the object structure of the chatbot project

The current version of Express.js framework does not include session management. The developers of Express.js apparently have decided to remove that from the standard library. For avoiding writing session management from scratch, adding the express-session package should be included. The express-session package generates a new session for each connection made through the socket. The session-id generated via the session package will be used as an identifier for the rest of the project, though a disconnect event through the socket will remove the session, which makes it forget past actions on every browser refresh. On code example 7, is the basic initialization of the session middleware.

```
// Init session middleware
var sessionMiddleware = session({
  secret: '48870281fbc147469d5d4b5078d4d9cf',
  resave: true,
  saveUninitialized: true,
  cookie: {
    maxAge: 60000
  }
});

app.use(sessionMiddleware);

// Add session middleware to Socket.IO
io.use(function (socket, next) {
  sessionMiddleware(socket.request, socket.request.res, next);
});
```

Code 7. A session middleware needs to be implemented for identifying users that connect to the socket.

Once the session has been created, it is necessary to identify each web socket connection for identifying the connection. This can be achieved by adding a session middleware to the socket. This way the socket contains the session identification and all the connections can be identified. No information is saved about the user after he/she disconnects from the application. This way the project will only contain information in memory. The configuration that was created simply gets removed from the list of configurations while the configuration in the CPQ-service gets saved.

Both Rivescript and the CPQ-handler use the session identifier generated by express-session package to remember the configuration and the conversation. Using an identifier with Rivescript makes it possible to remember the user's name if he/she decides to tell it to the chatbot among many other things.

4.3 The beginning of the session and quotation configuration rotation

Once the socket connection gets established between the client and the conversational agent, the conversational agent should make a greeting that the user will know that the connection was successful. Since a user might not know what the agent's purpose is, enabling the option to ask the chatbot about its features should be done via LUIS. After the greeting, the chatbot should wait for the user to respond.

If the encouragement to have dialog with the conversational agent draws the user's attention, the Rivescript brain will take care of everything that is under the scope of the conversational agent. In a case where a configuration is found from the user's session, the Rivescript brain should be skipped and the configuration should continue until it is either finished or the configuration is paused.

In a case where the configuration is paused, LUIS should be used for identifying if it should be started again. In a case where the user wants to start a fresh chatbot, the sessions can be reset by reloading the page. This should come in handy, since there can be uncounted issues that cause the chatbot to stop responding. This feature was built mostly for debugging and testing purposes.

The main reason for the chatbots existence is to provide a way to generate a quotation and download it when it's finished. Starting the quotation generation should be as easy as possible. Taking into account that the user might not know how the chatbot works, there should be multiple sentences that initiate the quoting process. This is the first phase, where LUIS will be helping to determine the intentions of the user. LUIS-service should take control of interpreting the user's intentions from the sent sentences to give an accurate approximation if the user indeed wanted to start the configuration. The thought process of the program is visualized as a diagram in figure 13.

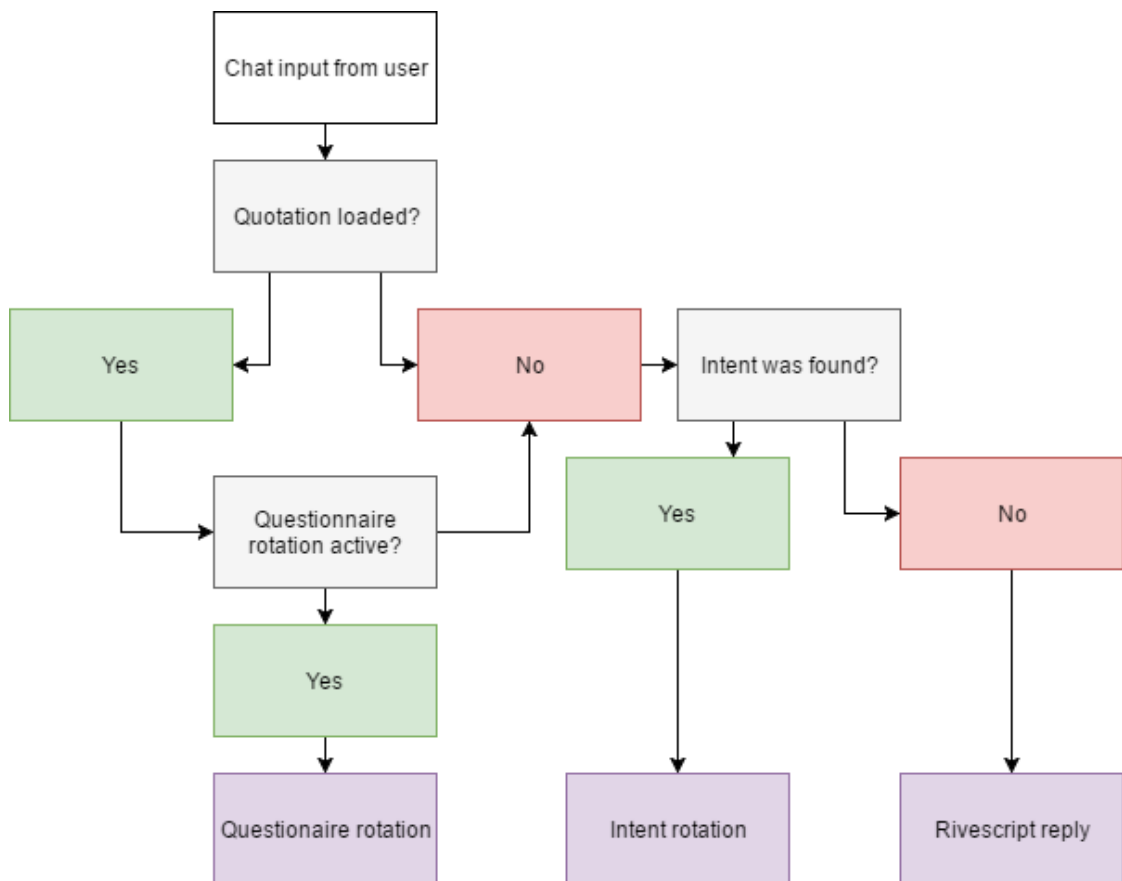


Figure 13. The chatbot has multiple states that provide different functionality, but the user needs to be in the correct state to use the feature.

After the configuration has been started, the chatbot should start asking questions to finish the quoting process. The amount of questions can be huge, so a pause/cancel option should be implemented, so that it would not be necessary to refresh the page every time something goes wrong.

4.4 Quotation creation workflow in the chatbot

A standard CPQ-project would contain a normal user interface with multiple pages to go through to finalize the proposal. The configuration fields in these pages are content that will be asked within the configuration process in the chatbot. These pages will be passed one-by-one through the custom-built API, thus making the CPQ workflow a bit different from the chatbot's quotation workflow.

The quotation configuration starts with a request from the user to either start a new proposal or load an existing one. The program validates the set of questions and continues

from the first question in the dataset that hasn't been answered yet. After all of the questions from the dataset has been answered, the program sends the values to the CPQ-service and returns the URL for the user to download the proposal. At this point the chatbots state returns back to the generic Rivescript, so that a normal dialog can occur. A visualization of how the configuration in the project can be seen in figure 14.

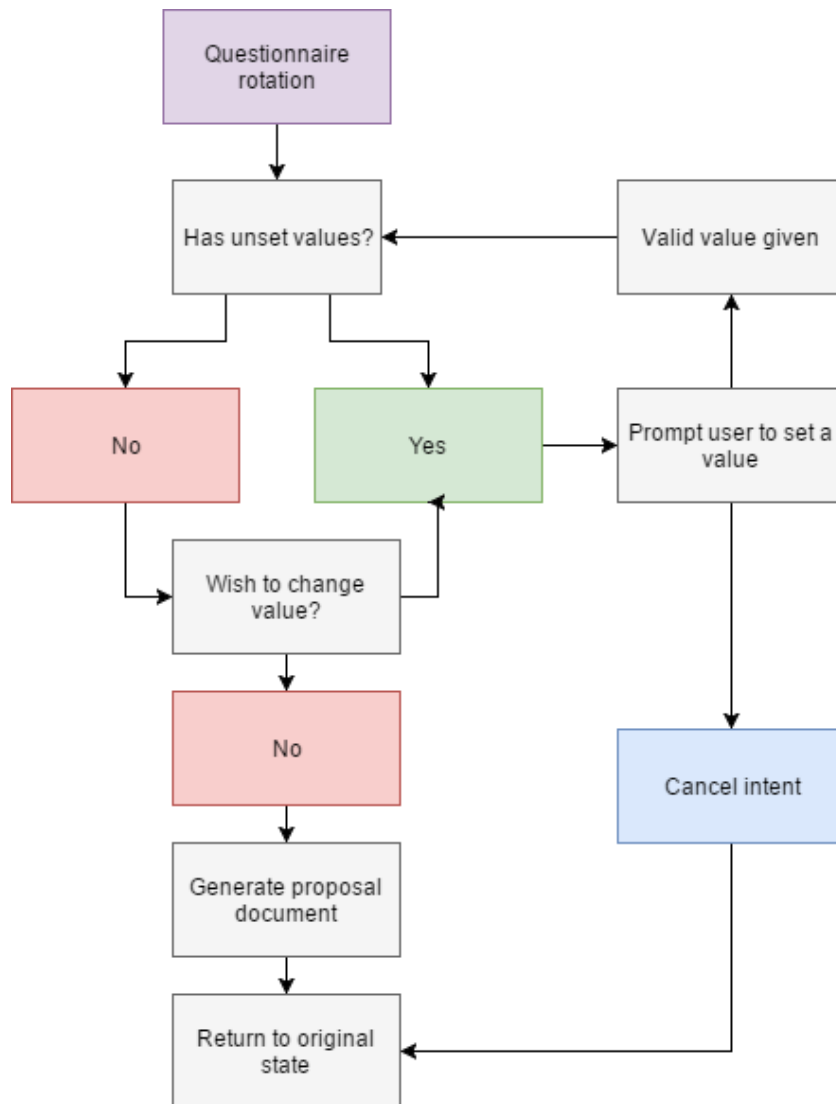


Figure 14. Rotation of the quotation creation workflow

The actual business logic happens inside the CPQ-project, so none of the data handling will be done in the chatbot project. The only thing the chatbot will do is that it takes the data and sends it to the CPQ-project. A basic CPQ API JSON array looks like following:

```
{
  "text_input": {
```

```

    "text": "A sample text input",
    "fieldName": "SampleTextInput"
  },
  "number_input": {
    "type": "number",
    "text": "A sample number input",
    "fieldName": "SampleNumberInput"
  },
  "dropdown_input": {
    "text": "A sample dropdown input",
    "type": "dropdown",
    "fieldName": "SampleDropdownInput",
    "options": {
      "1": "Option #1",
      "2": "Option #2",
      "3": "Option #3",
      "4": "Option #4",
      "5": "Option #5"
    }
  },
  "radio_input": {
    "text": "A sample Boolean input",
    "fieldName": "SampleBooleanInput"
  }
}

```

Code 8. JSON example of data, passed through the CPQ-project API

The CPQ-project's API informs about the questions and what type they are, thus making the chatbot only a medium for interaction without any specific logic behind it with an exception to showing the questions one after the other. The responses and requests happen with the same JSON array, defining same kind of values like the CPQ-project does in its own UI. Code snippet 9 is an example of the request the client can send to the project.

```

{
  "timeStamp": 1486382429823,
  "text": "2.2.2012",
  "server": false,
  "fieldName": null,
  "type": null
}

{
  "fieldName": "sampleInput",
  "value": null,
  "text": "Insert random text",
  "prompting": true
}

```


Code 9. JSON example data of between client and the chatbot

The JSON array gathered from the user and the API gets sent back to the CPQ-project where the proposal document can be generated. A link provided through the API makes it possible to download the generated proposal based on the quotation's configuration.

While the program is running, the command line informs the developer about current discussions by logging every connection/disconnection and every request/response that gets process through the chatbot. This data can be used for further development in the future and provide a debugging tool when someone is trying out this proof-of-concept and manages to break something in the agent. This approach makes it easy to trace back the user's steps like on dialog 1.

1. Rivescript for Configurations: #0 has finished loading!
2. Rivescript ChitChat: #0 has finished loading!
3. Socket: SESSION_ID connected.
4. Socket: SESSION_ID <= Server: Hello! I'm Chatbot. What do you want to do?
5. Socket: SESSION_ID => Server: Hello
6. Socket: SESSION_ID <= Server: Hi there!
7. Socket: SESSION_ID => Server: How are you?
8. Socket: SESSION_ID <= Server: I am fine, thank you.
9. Socket: SESSION_ID => Server: Bye
10. Socket: SESSION_ID <= Server: See you later undefined.
11. Socket: SESSION_ID disconnected.
12. QuotationHandler: User disconnected, but didn't make a quotation SESSION_ID.

Dialog 1. Example command prompt output when someone is using the service.

4.5 Intent workflow before entering the quotation configuration workflow

A CRUD should be implemented to control the quotation configurations through the chatbot. This will be achieved by teaching LUIS to recognize sentences that will be used as actions. These actions that LUIS will return from its API will be used to run different commands on the CPQ-software. These actions are:

- Create

- Load/Read
- Update
- Delete
- List Quotations
- List quotations values
- Generate proposal

Some of the features are not acceptable until a specified state in the configuration process has been activated. The figure 15 visualizes different states and how when the commands are accessible by the user.

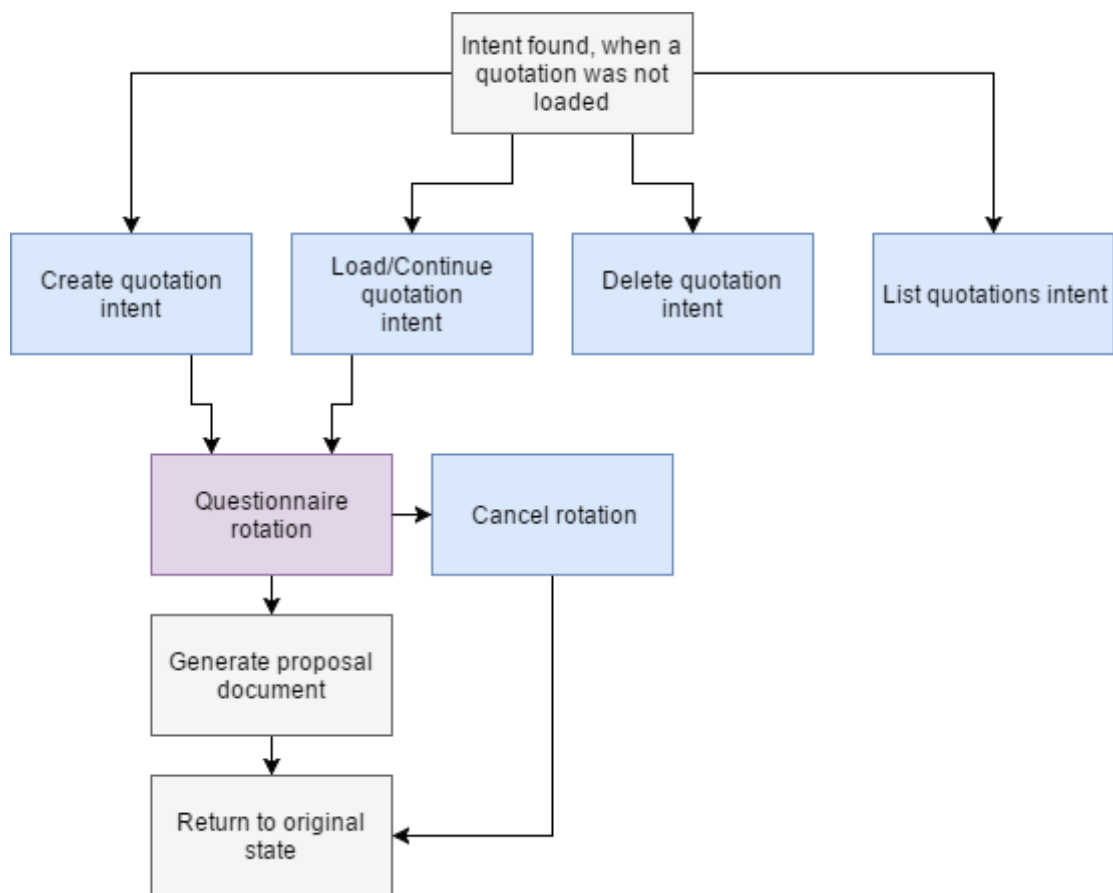


Figure 15. The intent can have multiple actions, though not all the intents are accessible until the program is in a specified part of the quotation configuration.

These states will be identified by checking the configuration saved in memory and identifying the current state of that configuration. If the session doesn't contain a configuration, then the commands available are: create, load and list. When a configuration is

found for the session, the configuration process can be continued, discarded or a new one can be loaded in its position.

5 Results

Using React and Node as the technology stack were excellent options for the proof-of-concept. The project provided a good way to get familiar with the newest trends in web development while scratching the surface of conversational agents. The drawback of selecting a new technology stack was that it took longer to get used to. Choosing LUIS happened as an accident, when studying possible options for understanding sentences through machine learning. Their examples and open source JavaScript library was something I tested the concept of intent recognition and it ended up staying in the project. If this proof-of-concept would be developed further in the future, it might be good idea to branch out of LUIS and try to use something else to detect the user's intentions. One of the options that I've been looking at is Google's open sourced Tensor flow.

In the original plan of creating the proof-of-concept, the chatbot was supposed to be only a closed domain command system that could not respond to random responses, making it only useful if the user would use commands instead of trying to have a dialog with the chatbot. While reading about chatbots I came across the key-value pairing pattern and decided to add one of the libraries to the project to provide more features to the chatbot, which in this case was Rivescript.

By design, the chatbot can only do simple value declaration when doing the configuration process. Thus, the chatbot is not ready to provide a solution as an equal medium to the user interface. The CPQ-project's fictional configuration is far simpler to what most of the real cases are. Though it would not be impossible to increase the number of features, the amount of work required to train LUIS and new logic for every configuration would take a large amount of time that would be a waste at the proof-of-concept level of development. Nevertheless, the amount of time taken to configure all pages through the user interface in the CPQ-project against the time it takes to run the configuration process in the chatbot is quite close, it might be safe to claim that the chatbot is marginally faster due to its design.

5.1 Evaluating the performance of conversational agents

In 1950 an English scientist called Alan Turing published a paper called “Computing machinery and intelligence”. Alan’s paper was one of the first papers that discussed about artificial intelligence and asked the question “Can machines think?”. He designed a game for figuring out if a machine could be capable of thinking, which is nowadays called the Turing test.

The Turing test has evolved to have multiple variations for different purposes, though the standard interpretation is still a viable test in the digital age. The test consists of three participants from which one is a computer. Just as in figure 16, player A (computer) has a discussion with player B (human), while player C (human) tries to determine which one of these two players is the computer. The spectator is also allowed to interrogate the other participants with predefined questions.

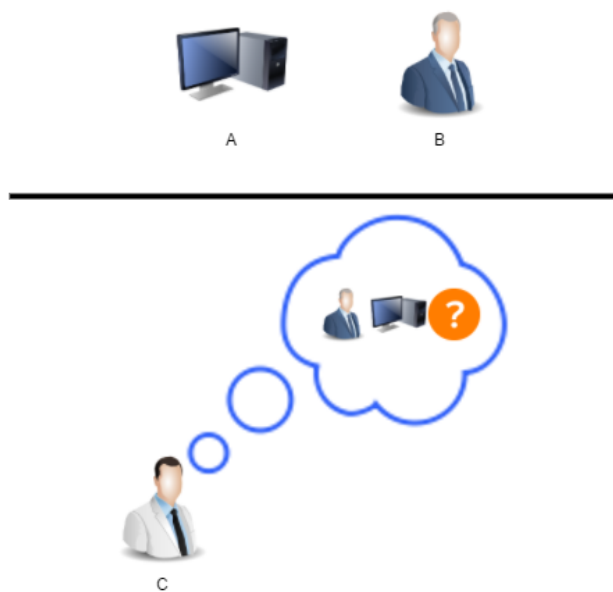


Figure 16. Visualization of the touring test.

An annual competition was introduced in 1990 by a man called Hugh Loebner. The competition, known as The Loebner prize promises the first team that passes the Turing test a 100,000\$ prize. [1] So far, no one has ever won the prize, though some good performances have tried. A Russian team created a conversational agent called Eugene Goostman that managed to fool 33% of the judges involved in the interrogation. This is a result that has never been achieved before. [24]

While the Turing's test is currently used as an indicator of current advancements in conversational agents, the fact that there isn't any requirement for artificial intelligence makes the test somewhat obsolete, since any kind of chatbot is viable to compete without any characteristics of intelligence. [23]

5.2 Rivescript evaluation

The reason for this project wasn't to create an intelligent conversational agent, so Rivescript should suffice for chitchat. Though it is interesting to use the basic Turing test fundamentals to test how well the dataset performs when the user is trying to make it break. Due to the complexity of modern chatbots it would've taken much more time and effort to delve deeper into the chatbot world and make it a bit smarter than what Rivescript provides. Rivescript's immersion can be easily broken after exchanging couple sentences. Rivescript gets easily lost when the sentence is used in the context of past discussion or the topic is very specific like politics. In dialog 2, there is an example of breaking immersion easily just by writing a random response.

13. Chatbot: Hello! I'm Chatbot. What do you want to do?
14. User: Hi!
15. Chatbot: Hi there!
16. User: What can you do?
17. Chatbot: I can do multiple things, but the most important task is creating and configuring a quotation while chatting with you about the specifics of the configuration.
18. User: How are you?
19. Chatbot: Everything is going extremely well.
20. User: Great!
21. Chatbot: Thanks for the compliment.

Dialog 2. Example of tricking the chatbot to reveal itself.

The simplicity of the dataset is noticeable on dialog 2 at the end, when Rivescript forgets the context. On dialog 3, it is noticeable that writing random topics and sentences can generate believable responses.

1. User: I'm Batman!
2. Chatbot: That seems like an interesting topic.

3. User: Who is Batman?
4. Chatbot: Batman is a fictional character who appeared first in a 1938 comic book.
5. User: Are you Batman?
6. Chatbot: You're not really talking about me – are you?
7. User: Yes.
8. Chatbot: That's interesting. Please continue.
9. User: My name is Fredrik.
10. Chatbot: That's my master's name too!
11. User: It is?
12. Chatbot: Sure, it is.
13. User: Where is he from?
14. Chatbot: Batman? I think he is from planet Earth.

Dialog 3. Example of having a random dialog with the Rivescript brain

The dataset used for Rivescript brain contains a lot of topics like the Batman discussion on dialog 3, though the immersion broke after entering another domain. Rivescript was still using Batman as it's topic after user had stated his name and drove the discussion to other domains.

In conclusion, Rivescript is not capable of holding a conversation with the user. The problem of understanding contexts and hoping for a result that makes sense requires immense amounts of data and a different approach for the way Rivescript works. The keyword pattern just doesn't seem to be able to provide sophisticated conversational agent. The current brain holds the basic AIML responses re-written for Rivescript and even the immense amount of data is not capable to uphold a believable conversation for long periods.

5.3 Testing LUIS

Since LUIS is crucial for controlling the CPQ-software, the identification of the utterances should be as good as possible. Testing how well the intent recognition works can only happen after the training process. In table 2 there is a set of test utterances that should be identified to actions and possibly all variations of these sentences. On some of the utterances there are parameters that are required for the utterance to work, like loading

a quotation requires the quotation identifier to be present in the utterance. With this approach we can verify that LUIS recognizes the important sentences for controlling the CPQ-software and would give an impression that it really does understand what the user wants to do.

Utterance	Intent/Action	parameters identified
I would like to create a quotation	create.quotation	no parameters
Could you create a quotation?	create.quotation	no parameters
Make a quotation for me.	create.quotation	no parameters
Create quotation, please.	create.quotation	no parameters
Load quotation Q219	load.quotation	quotation_identifier
Load a quotation with an ID of 219	load.quotation	number
Could you load a quotation for me? The ID is Q299	load.quotation	quotation_identifier
Please, load a quotation with the ID Q299	load.quotation	quotation_identifier
What quotations are there?	query.quotations	no parameters
Are there any quotations?	query.quotations	no parameters
Could I download the document?	query.document	no parameters
What are the values of the current quotation?	query.current_quotation	no parameters
Could you save the quotation?	save.quotation	no parameters

Table 2. LUIS utterance tests with actions recognized correctly

Since the intent recognition based on the test works well, there is no need to continue training LUIS. Extending LUIS with new intents is easy if the sentence does not contain too much data, thus making the creation of intents quick. Basically, the functionality can be vastly extended by training LUIS.

5.4 Testing configuration

The quotation process is the simplest part of the chatbot, where there is no added functionality from LUIS or Rlvescript. In this part, if a cancel intent has not been identified, the chatbot will only try to set the values into the configuration set-up. To add some “human like” response to the configuration process, the chatbot writes suffixes and prefixes to the questions such as in example dialog 4, which is a real conversation between the user and the chatbot.

1. You: Hi there

2. ChatBot: Hi there!
3. You: I would like to create a quotation
4. ChatBot: I see.. hmm. What's the project name?
5. You: Proof-of-concept project
6. ChatBot: Ah.. Could you give a value for application?
7. You: Testing purposes
8. /** Continuing with similar questions **/
9. ChatBot: Just a second... Select yes if you want to include warranty bank guarantee? (Yes/No)
10. You: Yes
11. ChatBot: Ah.. What would be the value for delivery time?
12. You: 123
13. ChatBot: I see.. Just a second.. I'll need a value for date shown on the documents?
14. You: 9.2.2017
15. ChatBot: Let me gather some information for the next step. What's the this quotation is valid until?
16. You: 9.3.2017
17. ChatBot: Let me gather some information for the next step. What would be the value for quotation sentiment?
18. You: Testing stuff
19. ChatBot: Hmm.. I see.. Just a second.. I need you to select one of the values from the dropdown below for payment terms template?
20. ChatBot: Let me see.. What would be the value for payment terms template?
21. You: test
22. ChatBot: Hmm... Select yes if you want to include warranty bank guarantee? (Yes/No)
23. ChatBot: Hmm.. Next step will be... I need to know if there should be a warranty bank guarantee? (Yes/No)
24. ChatBot: Let's see.. I'll need a value for signee?
25. You: Mr Fredrik Gundersen
26. ChatBot: Okay, there's nothing left to configure. I'll generate a file for you to see what has been done.
27. You: Could you save the quotation?
28. <<Quotation saved. A file provided for the user to download>>

Dialog 4. Example of doing the configuration with the chatbot.

The prefixes before the question are noticeable since every question follows the same structure. An example of the prefix could be something like on the row 24 where the

chatbot asks: “Let’s see... I’ll need a value for [VALUE]?”. The value is representing the keyword of what value should be set when the user answers the question.

After the configuration is complete, the chatbot saves the configured values to the CPQ-project and generates a link for downloading the proposal document. With the download link, there is no need to visit any extra page for generating the proposal document.

6 Summary

The amount of chatbot services has been steadily increasing, from Facebook’s messenger application to Microsoft’s Skype. This trend of conversing with an AI has provided tools and software to make it possible for anyone to create their own conversational agents with little experience in programming. In the past, chatbots were mostly for novelty or a tool most commonly found on chat rooms in the internet, but now they’ve emerged also on e-commerce.

There are several approaches for creating a chatbot, some of the programs use already sophisticated learning algorithms and language analysis tools. These algorithms can be used as intent recognition tools or to generate a taught response to whatever the user has written. With keyword patterns a chatbot can be implemented very quickly, though making it smarter takes a lot of effort, due to the number of keywords and new logic that need to be hand written by the creator. Neural networks and learning software might be one of the most interesting approaches, though they require a huge dataset to learn from before they can make reasonable responses.

The project was created to provide an alternative medium, such that would minimize the amount of interaction with a traditional form-styled user interface. A CPQ-software should be operated through the chatbot with its most simplistic features and possibly have novelty features when the CPQ-software was not being used. The chatbot was created as a proof-of-concept with free tools that are available to anyone.

By using a keyword library such as Rivescript or something similar, a set of predefined responses can be provided for responses that are not part of the CPQ-process, though the only way to make this kind of library have immense amount of responses would require a huge dataset that needs to be hand written by the developers.

Making the program understand the user's intentions better, an intent recognition tool like LUIS will be used. After training LUIS with example sentences, it can understand intentions and change them into actions. In addition to understanding intentions, LUIS can differentiate values and keywords from the sentence, so that they can be used in the software as parameters or attributes. With requirement for training the chatbot, the drawback of having huge datasets arrives again.

The API for the project was done with Node.js, using Express.js as the framework of choice for handling the requests. The chat uses sockets so that following connections would be easy and providing responses would be effortless. Rivescript is used to provide answers to things that would not be part of the CPQ-software features and LUIS would use its intent recognition features to understand when the CPQ-software supposed to be controlled. The client's user interface was created by using React.js that provided an easy way to render new messages into the application. The CPQ-software was a ready product provided by the company I was working for, modified to fit the basic functionalities that were needed by the project.

After having the Rivescript library as the novelty chatbot and using LUIS to understand intentions for using the CPQ-software, the project performed well in the scope it was bound in. The project could talk with the user with the predefined responses via Rivescript and understand answers that the user might give on chat-mode or configuration-mode. By training LUIS much further, it would be possible to use the whole software purely by telling the project what to do, but that would require a huge amount of data to train with and configuration, so that LUIS would understand all possible options and variables.

Sources

- 1 Bots as Language Learning Tools. 2006. Online document. Language Learning & Technology. <<http://lt.msu.edu/vol10num3/emerging/>>. Read 31.1.2017.
- 2 The complete beginners guide to chatbots. 2016. Online document. Chatbots Magazine. <<https://chatbotsmagazine.com/the-complete-beginner-s-guide-to-chatbots-8280b7b906ca#.zds2hpp2i>>. Read 31.1.2017.
- 3 Functionality provided by systems for synchronous conferencing. 2003. Online document. Carl Von Loesch. <<http://www.psyc.eu/synconf>>. Read 31.1.2017.
- 4 Welcome to the Slack API. 2017. Online document. Slack. <<https://api.slack.com/bot-users>>. Read 31.1.2017.
- 5 Fortune 500. 2017. Online document. Fortune. <<http://beta.fortune.com/fortune500>>. Read 31.1.2017.
- 6 Messenger Platform. 2017. Online document. Facebook, Ltd. <<https://developers.facebook.com/docs/messenger-platform>>. Read 31.1.2017.
- 7 Bots, Messenger and the future of customer service. Online document. Techcrunch<<https://techcrunch.com/2016/05/07/bots-messenger-and-the-future-of-customer-service/>>. Read 2.2.2017.
- 8 Why Facebook and Microsoft say chatbots are the talk of the town. Online document. The Guardian. <<https://www.theguardian.com/technology/2016/sep/18/chatbots-talk-town-interact-humans-technology-silicon-valley>>. Read 2.2.2017.
- 9 The future of chatbots. Online document. Business Insider. <<http://www.businessinsider.com/the-future-of-chatbots-2016-12?r=US&IR=T&IR=T>>. Read 2.2.2017.
- 10 About Node.js. Online document. Node.js foundation. <<https://nodejs.org/en/about/>>. Read 2.2.2017.
- 11 What is NPM?. Online document. NPM, Inc. <<https://docs.npmjs.com/getting-started/what-is-npm>>. Read 2.2.2017.
- 12 A package manager for the web. Online document. Twitter. <<https://bower.io/>>. Read 2.2.2017.
- 13 git. Online document. <<https://git-scm.com/docs/git>>. Read 2.2.2017
- 14 Express.js. Online document. <<http://expressjs.com/>>. Read 2.2.2017.

- 15 klaroCPQ. Online document. Sujuwa Ltd. <<http://klarocpq.fi/>>. Read 2.2.2017.
- 16 AIML: Artificial Intelligence Markup Language. Online document. A.L.I.C.E AI Foundation Inc<<http://www.alicebot.org/aiml.html>>. Read 2.2.2017.
- 17 Rivescript. Online document. <<https://www.rivescript.com/>>. Read 2.2.2017.
- 18 Microsoft deletes racist genocidal tweets from AI chatbot Tay. Online document. Business Insider Inc. <<http://www.businessinsider.com/microsoft-deletes-racist-genocidal-tweets-from-ai-chatbot-tay-2016-3?r=US&IR=T&IR=T>>. Read 2.2.2017.
- 19 Understanding natural language. Online document. Microsoft Ltd. <<https://docs.botframework.com/en-us/node/builder/guides/understanding-natural-language/>>. Read 2.2.2017.
- 20 LUIS-API. Online document. Microsoft Ltd. <<https://www.microsoft.com/cognitive-services/en-us/LUIS-api/documentation/home>>. Read 2.2.2017.
- 21 React. Online document. Facebook Inc. <<https://facebook.github.io/react/>>. Read 2.2.2017.
- 22 Cleverbot computer chats human. Online document. Live Science <<http://www.livescience.com/15940-cleverbot-computer-chats-human.html>>. Read.16.2.2017
- 23 Turing. Online document. University of Toronto <<http://www.psych.utoronto.ca/users/reingold/courses/ai/turing.html>>. Read 2.2.2017
- 24 Turing tested: An interview with Eugene Goostman, the first computer programme to pass for human. Online document. Independent Print Limited. <<http://www.independent.co.uk/life-style/gadgets-and-tech/news/turing-tested-an-interview-with-eugene-goostman-the-first-computer-programme-to-pass-for-human-9535740.html>>. Read 2.2.2017
- 25 H&M chatbot screen capture. Image. Chatbots Magazine. <https://cdn-images-1.medium.com/max/1600/1*dtY7dJkP5DQdf6ZRH06Mg.png>. Read 1.3.2018
- 26 Amazon Alexa. Image. Amazon Ltd. <http://media.corporate-ir.net/media_files/IROL/17/176060/EchoDot/Echo%20Dot-Black,%20Low%20Angle,%20On.jpg>. Read 1.3.2018
- 27 NLP tree. Image. NLTK org. <http://www.nltk.org/_images/tree.gif>. Read 1.3.2018